System Design Document for android application DrinkIT

Kajsa Bjäräng, Viktoria Enderstein, Elin Eriksson, Lisa Fahlbeck, Alice Olsson
21/10-2018
version 3

1 Introduction

DrinkIT is a pre party game application designed for groups of two to ten players. The game revolves around interaction between players and the games purpose is to create fellowship for new as well as old friends. By completing the challenges DrinkIT gives you, you will be ready to take over the night with your now closer friends.

1.1 Design goals

The goal for the design of the application is to create a functional game with few dependencies, that is easy to reform and develop due to loose coupling. All classes should, to the greatest possible extent, have only one responsibility and not interfere with classes they have nothing to do with. The classes should thereby be isolated and encapsulated.

1.2 Definitions, acronyms, and abbreviations

DrinkIT - The name of the application as well as the name of the main class in the model package.
Challenge - A task to perform for each player when it is their turn, for example answer a question or perform a song or charade.
Category - All challenges are divided into one of nine different categories.
GameRound - GameRound contains all information needed to distinguish and save statistics about any round during the game. It contains a player, a challenge, and whether the challenge was suceeded or failed.
Gradle - An open-source build automation tool based on Groovy and Kootlin.
Travis - Automatically builds the code and sees if everything still works after a commit to GitHub. Otherwise it report that something went wrong, which is good when several people work with the same project.
Android Studio - An integrated development environment (IDE). It’s the official IDE for Google’s Android Operating system.

2 System architecture

The application is made in Android Studio version 3.1.4, written in the language Java. It uses the built in automation tool Gradle version 4.4 which comes included in this version of Android Studio. The application is designed to be played on a single unit of an android smartphone.

2.1 Game flow

To start the game the user has to add all participating players by name, then choose which categories should be included and the duration of the game. Thereafter a challenge from one of the chosen categories is randomly shown on the screen. Most challenges are aimed to a specific player, whos name will then show up together with the challenge. Other challenges are aimed for the whole group of players. Some challenges gives the player points if they are accomplished whilst others are just for fun. After the number of challenges corresponding to the chosen duration have been played the game is finished and a scoreboard is shown with an option to continue playing. During the game there are options always available to add another player, remove an existing player, quite the game or access instructions and help.

3 System design

The application is designed to make it easy for future developers to understand, navigate and expand the code. To achieve this the development of the application is built on a number of design principles and patterns.

3.1.1 Single responsibility principle

Classes doing one thing only, and doing it well, leads to a robust code. In the application DrinkIT one of the goals was to make sure all of the classes and class methods follows the Single Responsibility Principle. This has so far been implemented partially throughout the code.

3.1.2 Open Closed Principle

Making the application open for extension but closed for modification is achieved by the implementations of design patterns such as the Factory Pattern and the Model-View-Controller Pattern. These patterns will make it easy for the future developers to expand DrinkIT while also making it difficult to modify the code.

3.1.3 Encapsulation

The goal of encapsulation is to avoid global access of variables and methods which prevents data to be modified by an outside user. This leads to a more legible code which is easier to expand, and has been implemented in the code.

3.1.4 Factory pattern

Following the Factory pattern helps to hide internal implementation and reduces dependencies and code duplication, which in turn leads to code that is easier to read and understand.
DrinkIT has two factories, ChallengeFactory and CategoryFactory both located in the model package.

The main reason to implement the use of the Factory Pattern in the application comes from the need for incapsulation and hiding of internal implementation. This is why the Challenge class has been made abstract, and it's ChallengeFactory hides the creation of it's different subclasses.

The CategoryFactory works slightly differently as Category does not have subclasses to hide, but comes to good use as there are several categories to create each time the game is played.

3.1.5 Model-View-Controller Pattern (MVC)

MVC is a pattern usually used when an application contains some type of view. The goal of the dependency is to achieve as high cohesion and as low coupling as possible. Due to DrinkITs many views and gamelogic the implementation of the MVC pattern was natural.

The model of the application is in the package Model where the Class DrinkIT is the heart of the model. The View contains multiple Activities which lie in the View package and the Controller class lies in the Controller package and acts as a connection between the Model and View packages.

The model is responsible for the domain logic and holds several classes with different responsibilities. The model consists of ten classes whereof one is an abstract class, Challenge.

The class DrinkIT acts as an interface to the Model package from the Controller. It's the Controller which interacts with the players since it takes input from the different views and calls for the right methods from the model.

3.2 Design model

The design model consists of three different packages, View, Model and Controller and follows, as earlier mentioned, the design pattern MVC. The class DrinkIT which lies in the Model package, holds a key position since DrinkIT is the gate between the

Controller and the Modell. It is possible to say that the DrinkIT class is the heart of the game since it holds almost all of the underlying game logic.

MainAppActivity initializse the game and creates an instance of each package which is used throughout the game. MainView is created in MainAppActivity to decrease the dependencies by acting as a superclass to all other views, thereby holding the only instace of the Controller available to all Views.

When starting the game the startpage is a simple view with two buttons to make it a clear entry point to the game. When starting the game the user needs to add all players by manually entering their names. This view is designed so that the user dynamically gets error messages if the input entered is incompatible, for example duplicate names of players. The game is developed accept a minimum of two players and no duplicate names. There is currently a known glitch that the user needs to enter the same name and one extra letter, then remove the last one before the error messages work. This is naturally to be improved in the future.

DrinkIT contains a list of players. The class Player is the type of player added and contains a name and a point. The players are added in the list in the beginning of the game as explained above. This list keeps track of the players and their points during the game by being used in different methods in DrinkIT.

DrinkIT also contains a list of categories. The class Category contains a name, an instruction on how to play the Category in question, a list of challenges, the indexOfActiveChallenge and a state, either active or inactive.

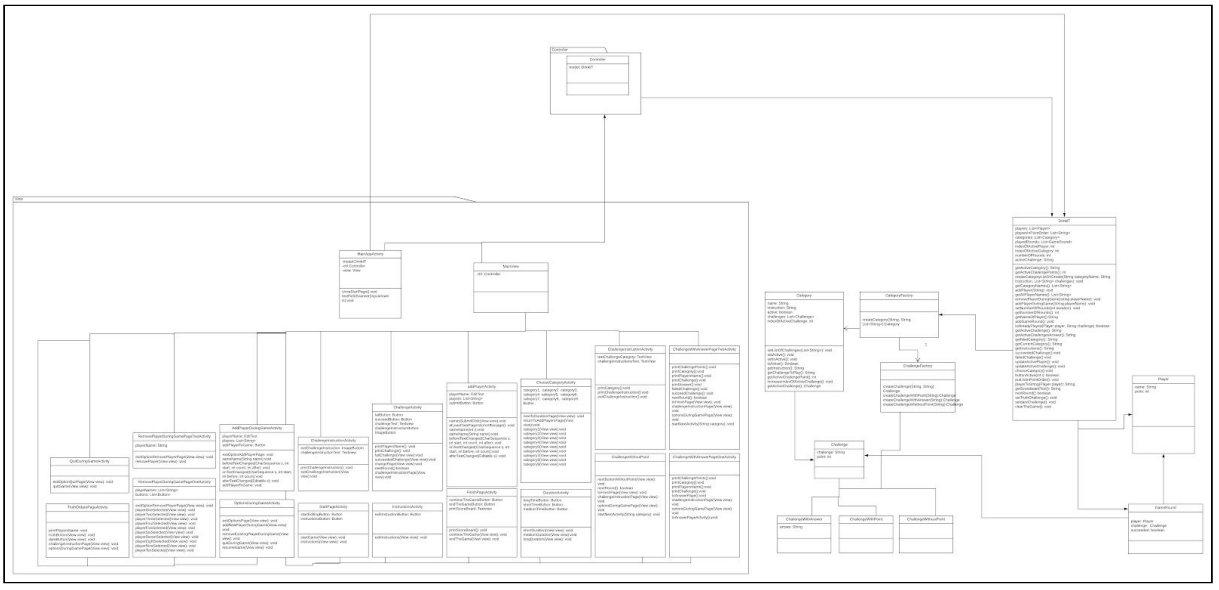
The user gets to choose from nine categories which they would like to include in this instace of the game. When selecting a category, its state is set to active from the default of inactive. DrinkIT contains an integer indexOfActiveCategory which is looped through each gameround to help in the selection of which challenge to show next.

Each category contains a list of Challenges that belong to that category. All challenges are however not the same, which is why the class Challenge has subclasses which are configured differently. The subclasses are ChallengeWith Answer, ChallengeWithoutPoint and ChallengeWithPoint. Therefore, as explained earlier, a factory is used to create the challenges differently according to which category they belong to. Further on in the text, under *4. Persistent data management*, there can be found an explanation of how challenges are fetched to the view when they are to be shown to the user.

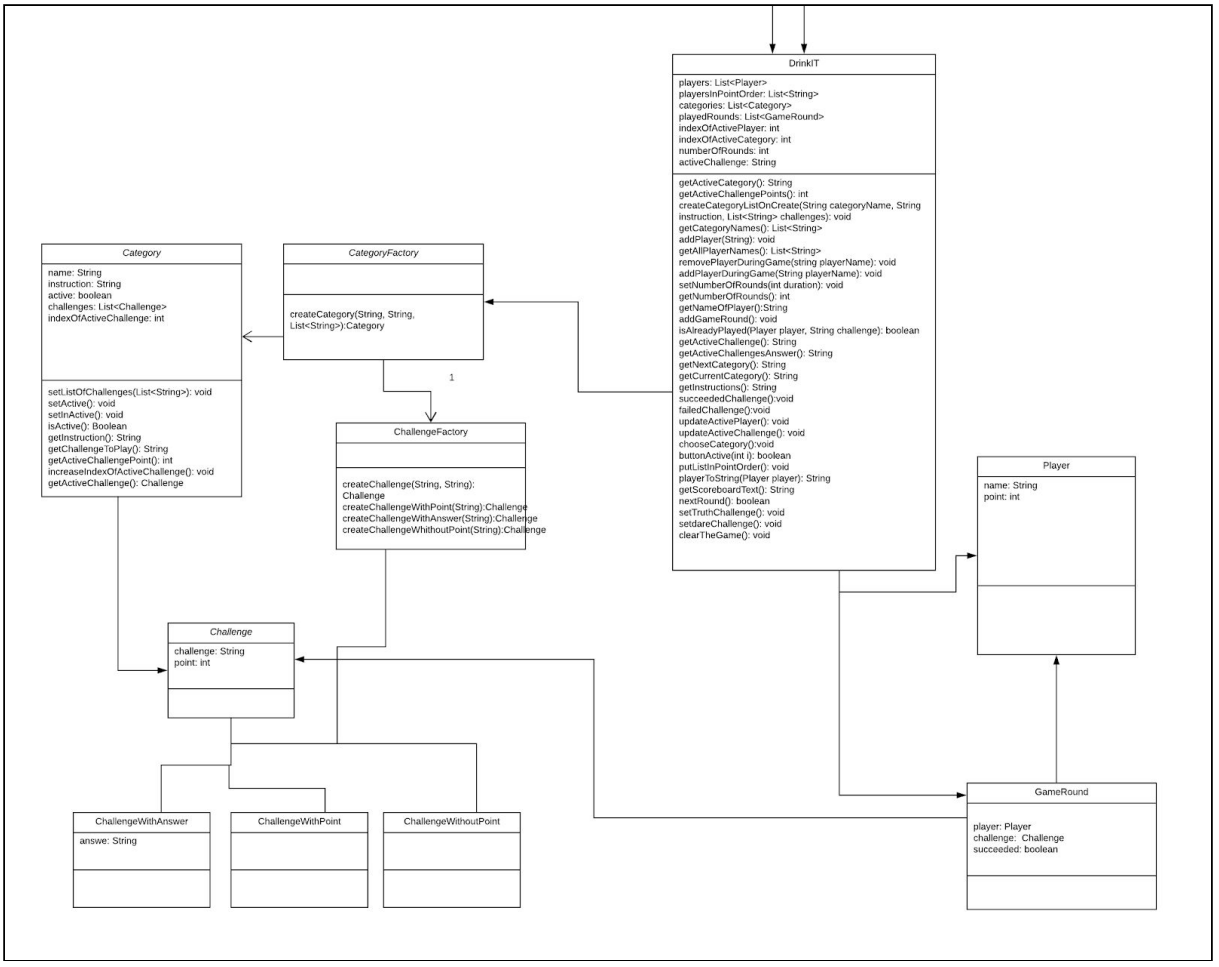
GameRound is the class that assists the application in keeping track of all statistics of earlier played rounds. This is done by saving each played GameRound in a list in DrinkIT, and is used to for example make sure the same player never has to play the same challenge twice.

To loop through the players and ensure each player gets to play the same number of rounds during the game, DrinkIT contains an integer, indexOfActivePlayer, which is incremented after each completed round. Every time the index reaches the last player in the list, the list is shuffled and the index is set back to 0. This is to avoid players consistently showing up in the same order, which we believe make the game more interesting to play. There is also logic to prevent the same player showing up twice in a row, even when the list is shuffled and the index reset.

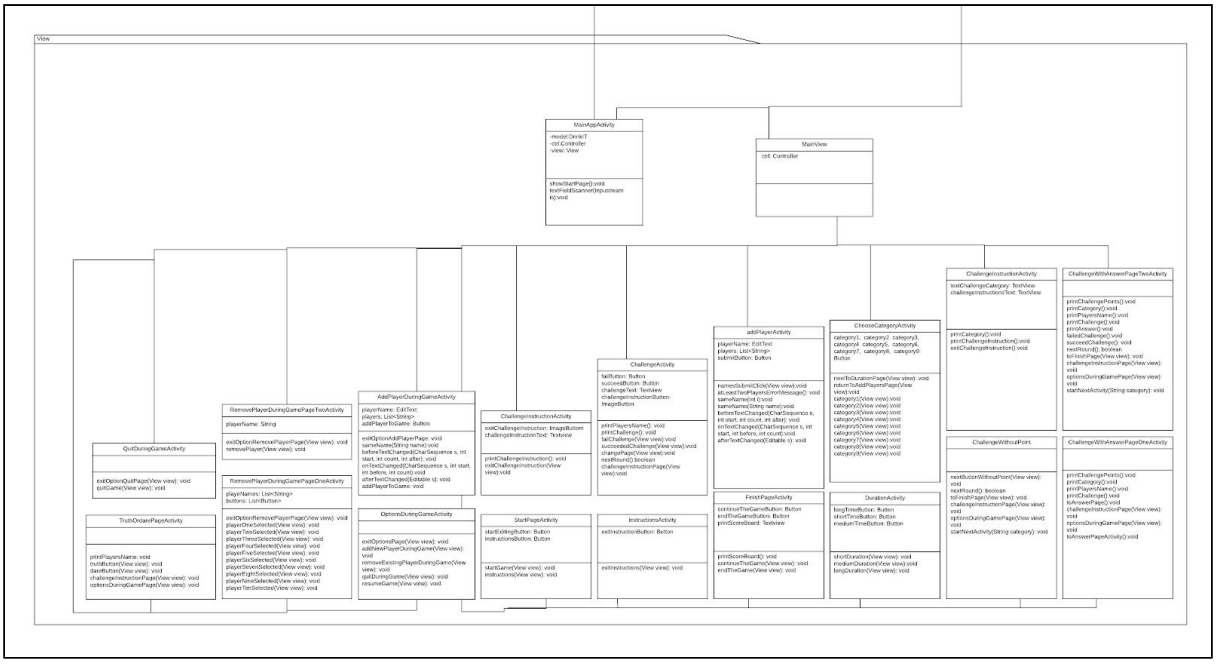
The Design Model:



The Model package in more detail:



The View package in more detail:

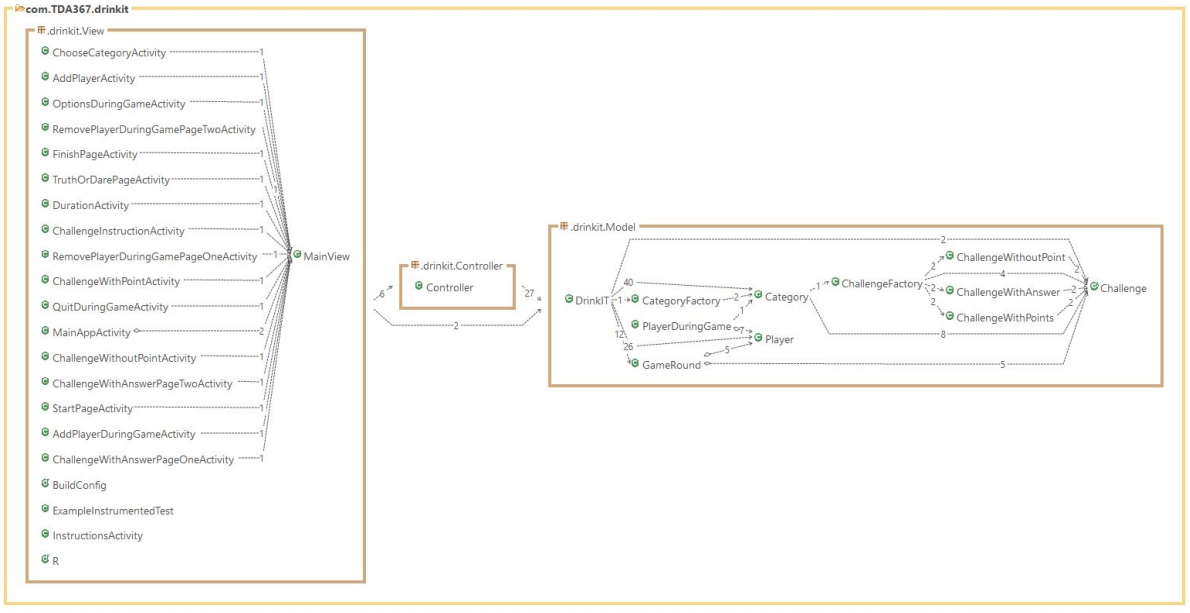


3.3 Quality (test) //eller egen punkt 4

During the implementation of the application, tests have been implemented after every new solves user story, to make sure the code that has been written is solid and works correctly. The application contains a test package, com.TDA316.drinkit (test), which contains multiple unit tests, made with the framework JUnit. Which is located in the java package of the code.

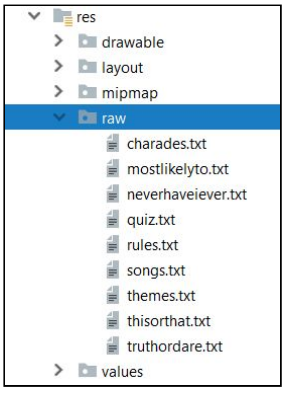
3.4 Diagrams

Dependency analysis



4 Persistent data management

Textfiles are used for all categories in the game. Each category has its own textfile that contains all information about the challenges that belong to that specific category. The textfiles



can be found in the folder “raw”, which in turn can be found in the folder “res”.

When the application starts, all textfiles get scanned in the class *MainAppActivity* and the categories are created. The textfiles gets scanned from the top down. The first row is what will be set as the categorys name. The second row in the textfield is what will become the instruction for that category.

Each nextcoming row in the textfiles are challenges. There rows are first made into a list of Strings. Now there are two of three things required to create a Category, a name and an instruction, however a Category requires a list of Challenges, and not a list of Strings. The list of strings is therefore sent to ChallengeFactory to become a list of Challenges. There are three different types of challenges and depending on the category differnet ones will be created.

An example of how the Strings of challenges are separated in the ChallengeFactory into these different parts is with the category Quiz. A quiz-challenge should contain a question, an answer and a point-value. Row three and down correspond to the Strings to be made into Challenges. The question and answer are separated by a “-” and the String is then searched and any number is picked out as the point-value.

```
Quiz
Quiz, write instructions here
Which three movies have received the highest number of Oscars? When you've given your answer, head to the
What is Chandler Bings job? When you've given your answer, head to the next page to see the correct answe
How many countries are there in the world? - 195 countries 4
Who gave the famous speech "I have a Dream"? - Martin Luther King 3
What was Ada Lovelace? - She was a mathematician, and is often regarded as the world first programmer 6
Who was the first female nobel prize winner? - Marie Curie 5
```

5 References

Gradle information:
http://www.vogella.com/tutorials/Gradle/article.html